Learning Phonotactics Using ILP

Stasinos Konstantopoulos
Alfa-Informatica, Rijks*universiteit* Groningen
konstant@let.rug.nl

February 5, 2008

Abstract

This paper describes experiments on learning Dutch phonotactic rules using Inductive Logic Programming, a machine learning discipline based on inductive logical operators. Two different ways of approaching the problem are experimented with, and compared against each other as well as with related work on the task. The results show a direct correspondence between the quality and informedness of the background knowledge and the constructed theory, demonstrating the ability of ILP to take good advantage of the prior domain knowledge available. Further research is outlined.

1 Introduction

The *Phonotactics* of a given language is the set of rules that identifies what sequences of phonemes constitute a possible word in that language. The problem can be broken down to the *syllable structure* (i.e. what sequences of phonemes constitute a possible syllable) and the processes that take place at the syllable boundaries (e.g. assimilation).

Previous work on the syllable structure of Dutch includes hand-crafted models [9, 1], but also the application of machine-learning approaches such as abduction [8] and neural networks [7, 6, ch. 4].

This paper describes experiments on the task of constructing from examples a Horn Clause model of Dutch monosyllabic words. The reason for restricting the domain is to avoid the added complexity of handling syllable boundary phonological processes. Furthermore by not using polysyllables no prior commitment is made to any one particular syllabification theory.

The machine learning algorithm used is described in Section 2, and then the paper proceeds to describe how the training examples are extracted from the corpus (Section 3), what prior knowledge is employed (Section 4) and, finally, draw conclusions and outline further research (Section 5).

2 Inductive Logic Programming and Aleph

Inductive Logic Programming (ILP) is a machine learning discipline. Its objective is the development of algorithms that construct Predicate Logic hypotheses that can explain a set of empirical data or observations.

The core operator used in ILP is *induction*. Induction can be seen as the inverse of deduction. For example from the clauses 'All humans die' and 'Socrates is human' the fact that 'Socrates will die' can be deduced. Inversely, induction uses background knowledge (e.g. 'Socrates is human') and a set of observations (training data) (e.g. 'Socrates died') to search for a hypothesis that, in conjunction with the background knowledge, can deduce the data. In more formal terms, given a logic program B modelling the background knowledge and a set of ground terms D representing the training data, ILP constructs a logic program H, such that $B \wedge H \vDash D$.

If the deductive operator used is resolution (as defined by Robinson [4]), then the inductive operator necessary to solve the equation above is the inverse resolution operator, as defined by Muggleton and De Raedt [3].

Aleph [5] is an ILP system implementing the Progol algorithm [2]. This algorithm allows for single-predicate learning only, without background theory revision or predicate invention. It incrementally constructs the clauses of a single-predicate hypothesis that describes the data, by iterating through the following basic algorithm:

- Saturation: pick a positive example from the training data and construct the most specific, non-ground clause that entails it. This is done by repeated application of the inverse resolution operator on the example, until a clause is derived that covers the original ground positive example and none other. This maximally specific clause is called the *bottom clause.*
- Reduction: search between the maximally general, empty-bodied clause and the maximally specific bottom clause for a 'good' clause. The space between the empty clause and the bottom clause is partially ordered by θ-subsumption, and the search proceeds along the lattice defined by this partial ordering. The 'goodness' of each clause encountered along the search path is evaluated by an *evaluation function.*
- Cover removal: add the new clause to the theory and remove all examples covered by it from the dataset.
- Repeat until all positive examples are covered.

The *evaluation function* quantifies the usefulness of each clause constructed during the search and should be chosen so that it balances between overfitting (i.e. covering the data too tightly and making no generalisations

that will yield coverage over unseen data) and overgeneralising (i.e. covering the data too loosely and accepting too many negatives). It can be simple coverage (number of positive minus number of negative examples covered by the clause) or the Bayes probability that the clause is correct given the data (for learning from positive data only) or any function of the numbers of examples covered and the length of the clause (for implementing bias towards shorter clauses).

Syntactic bias is applied during the reduction step to either prune paths that are already known to not yield a winning clause, or to enforce restrictions on the constructed theory, for example conformance to a theoretical framework.

3 Training Examples

As a starting point, a rough template matching all syllables is assumed. This template is $\mathcal{C}_3\mathcal{V}\mathcal{C}_5$, where $\mathcal{C}_n$ represents any consonant cluster of length up to n and $\mathcal{V}$ any vowel or diphthong. The problem can now be reformulated as a single-predicate learning task where the target theory is one of acceptable prefixes to a given vowel and partial consonant cluster. The rules for prevocalic and postvocalic affixing are induced in two separate learning sessions.

The underlying assumption in formulating the problem in this fashion is that the prevocalic and postvocalic material of a syllable is independent from each other. In other words, it is assumed that if a consonant cluster is acceptable as the prevocalic (postvocalic) material of a syllable, then it is acceptable as the prevocalic (postvocalic) material of all syllables with the same vowel.[1]

The training data consists of 5095 monosyllabic words found in the Dutch section of the CELEX Lexical Database, with an additional 597 reserved for evaluation. As has been mentioned above, however, the learning task is formulated as one of learning valid affixes to a partial syllable. In other words, the CELEX data is used to generate the actual training examples, which are instances of a predicate matching syllable fragments with phonemes that can be affixed at that point.

The positive examples are constructed by breaking the phonetic transcriptions down to three parts: a prevocalic and a postvocalic consonant cluster (consisting of zero or more consonants) and a vowel or diphthong. The consonant clusters are treated as 'affixes' to the vowel, so that syllables are constructed by repeatedly affixing consonants, if the *context* (the vowel and the pre- or post-vocalic material that has been already affixed) allows it. So, for example, from the word /maːkt/ the following positives would be gen-

[1] This assumption is consistent with hand-crafted syllabic models, although no claims are made here regarding its universality or prior necessity.

erated:

```
prefix( m, [], [a,:]  ).
prefix( ^, [m], [a,:]  ).

suffix( k, [],[:,a] ).
suffix( t, [k], [:,a] ).
suffix( ^, [tk], [:,a] ).
```

So, for example, the first two `suffix` rules should be read as follows: '/k/ can be suffixed to the /aː/ nucleus' and '/t/ can be suffixed to an /aːk/ syllable fragment'.

Note that the context lists in suffix rules are reversed, so that the two processes are exactly symmetrical and can use the same background predicates.

The caret, `^`, is used to mark the beginning and end of a word. The reason that the affix termination needs to be explicitly licensed is so that it is not assumed by the experiment's setup that all partial sub-affixes of a valid affix are necessarily valid as well.

In Dutch, for example, a monosyllable with a short vowel has to be closed, which means that the null suffix is not valid. The end-of-word mark allows for this to be expressible as a theory that does not have the following clause: `suffix( ^, [], [V] ).`

The positives are, then, all the prefixes and suffixes that must be allowed in context, so that all the monosyllables in the training data can be constructed: 11067 and 10969 instances of 1428 and 1653 unique examples, respectively.

The negative data is randomly generated words that match the $\mathcal{C}_3\mathcal{V}\mathcal{C}_5$ template and do not appear as positives. The random generator is designed so that the number of examples at each affix length are balanced, in order to avoid having the large numbers of long, uninteresting sequences overwhelm the shorter, more interesting ones.

The negative data is also split into evaluation and training data, and the negative examples are derived from the training negative data by the following deductive algorithm:

1. For each example, find the maximal substring that is provable by the positive `prefix/3` and `suffix/3` clauses in training data. So, for example, for /mtratk/ it would be `trat` and for /mlat/, `lat^`.

2. Choose the clause that should be a negative example, so that this word is not accepted by the target theory. Pick the inner-most one on each side, i.e. the one immediately applicable to the maximal substring computed above. For /mlat/ that would be `suffix(m,[l],[a])`. /mtratk/, however, could be negative because either `prefix(m,[tr],[a])` or `suffix(k, [t], [a])` are unacceptable. In such cases, pick one at random. This is bound to introduce false negatives, but no alternative that does not presuppose at least part of the solution could be devised.

3. Iterate, until enough negative examples have been generated to dis-

prove all the words in the negative training data.

One advantage of deriving the negative examples in this fashion is that the importance of examples 'closer' to the vowel is emphasised. This allows for a more balanced distribution of negatives along the possible consonant cluster sizes, which would have otherwise been flooded by the (more numerous) longer clusters.

4 The Background Theory

Since the problem is, in effect, that of identifying the sets of consonants that may be prefixed or suffixed to a partially constructed monosyllable, the clauses of the target predicate must have a means of referring to various subsets of $\mathcal{C}$ and $\mathcal{V}$ in a meaningful and intuitive way. This is achieved by defining a (possibly hierarchical,) linguistically motivated partitioning of $\mathcal{C}$ and $\mathcal{V}$. Each partition can then be referred to as a feature-value pair, for example LAB+ to denote the set of the labials or VOIC+ for the set of voiced consonants. Intersections of these basic sets can then be easily referred to by feature-value vectors; the intersection, for example, of the labials and the voiced consonants (i.e. the voiced labials) is the feature-value vector [VOIC+,LAB+].

The *background knowledge* is, as seen in section 2, playing a decisive role in the quality of the constructed theory, by implementing the theoretic framework to which the search for a solution will be confined. In more concrete terms, the background predicates are the building blocks that will be used for the construction of the hypothesis' clauses and they must be defining all the relations necessary to discover an interesting hypothesis.

For the purposes of this task, they have been defined as relations between individual phones and feature values, e.g. `labial(m,+)` or `voiced(m,+)`. Feature-value vectors can then be expressed as conjunctions like, for example, `labial(C,+)` $\wedge$ `voiced(C,+)` to mean the voiced labials.

Except for the linguistic features predicates, the background knowledge also contained the `head/2` and `rest/2` list access predicates. This approach was chosen over direct list access with the `nth/3` predicate, as bias towards rules with more local context dependencies.

The theories described in sections 4.1, 4.2 and 4.3 below, are based on background knowledge that encodes increasingly more information about Dutch phonology as well as Dutch phonotactics: for the experiment in 4.1 the learner has access to the way the various symbols are arranged in the International Phonetic Alphabet (IPA), whereas for the experiment in 4.2 a classification that is sensitive to Dutch phonological processes was chosen. And, finally, in section 4.3 a scalar[2] sonority feature is implemented, which

[2] As opposed to binary.

has been proposed with the explicit purpose of solving the problem of Dutch syllable structure.

The quantitative evaluation shown for three experiments was done using the 597 words and the part of the randomly generated negative data that have been reserved for this purpose.

4.1 The IPA segment space

For the first experiment the background knowledge reflects the way that the International Phonetic Alphabet (IPA, 1993 version) is organised: the phonetic inventory of Dutch consists of two disjoint spaces, one of consonants and one of vowels, with three and four orthogonal dimensions of differentiation respectively.

The consonant space varies in *place* and *manner of articulation,* and *voicing.* The manner of articulation can be *plosive, nasal, lateral approximant, trill, fricative* or *approximant.* The place can be *bilabial, alveolar, velar, labiodental, postalveolar* or *palatal.* Voicing can be present or not.

Similarly for vowels, where there are four dimensions: place (front, centre, back) and manner of articulation (open, mid-open, mid-closed, closed), length and roundedness.

The end-of-word mark has no phonological features whatsoever and it does not belong to any of the partitions of either $\mathcal{C}$ or $\mathcal{V}$.

This schema was implemented as one background predicate per dimension relating each phone with its value along that dimension, for example:

```
manner( plosive, p ).   place( bilabial, p ).
manner( nasal, m ).
```

The evaluation function used was the *Laplace function* $\frac{P+1}{P+N+2}$, where P and N is the coverage of positive and negative examples, respectively.

Since the randomly generated negatives must also contain false negatives, it cannot be expected that even a good theory will fit it perfectly. In order to avoid overfitting, the learning algorithm was set to only require an accuracy of 85% over the training data.

The resulting hypothesis consisted of 199 prefix and 147 suffix clauses and achieved a recall rate of 99.3% with 89.4% precision. All the false negatives were rejected because they couldn't get their prevocalic material licensed, typically because it only appears in a handful of loan words. The /ʤ/ prefix necessary to accept 'jeep' and 'junk', for example, was not permitted and so these two words were rejected.

The most generic rules found were:

```
prefix(A,B,C) :- A= '^'.
prefix(A,[],C).
```

```
suffix(A,B,C) :-  A= '^'.
suffix(A,[],C).
```

meaning that (a) the inner-most consonant can be anything, and (b) all sub-prefixes (-suffixes) of a valid prefix (suffix) are also valid.

Other interesting rules include pairs like these two:

```
prefix(A,B,C) :-
   head(B,D), manner(trill,D), head(C,E), length(short,E),
   manner(closed,E), manner(plosive,A).
prefix(A,B,C) :-
   head(B,D), manner(trill,D), head(C,E), length(short,E),
   manner(open_mid,E), manner(plosive,A).
```

and

```
prefix(A,B,C) :-
   head(B,D), manner(approx,D), head(C,E), length(short,E),
   place(front,E), voiced(minus,A).
prefix(A,B,C) :-
   head(B,D), manner(approx,D), head(C,E), length(short,E),
   place(front,E), manner(plosive,A), place(alveolar,A).
```

that could have been collapsed if a richer feature system would include features like 'closed or mid-open vowel' and 'devoiced consonant or plosive alveolar', respectively. These particular disjunctions might be unintuitive or even impossible to independently motivate, but they do suggest that a redundant feature set might allow for more interesting theories than the minimal, orthogonal one used for this experiment. This is particularly true for a system like Aleph, that performs no predicate invention or background theory revision.

4.2 Feature Classes

The second experiment a richer (but more language-specific) background knowledge was made available to the inductive algorithm, by implementing the feature hierarchy suggested by Booij [1, ch. 2] and shown in figure 1.

The most generic features are the *major class features* (CONSONANT and SONORANT) that are placed on the root node and divide the segment space in vowels [CONS-,SON+], obstruents [CONS+,SON-] and sonorant consonants [CONS+,SON+]. Since all vowels are sonorous, [CONS-,SON-] is an invalid combination.

The features specifying the continuants, nasals and the lateral /l/ are positioned directly under the root node, with the rest of the features bundled together under two *feature classes,* those of the *laryngeal* and the *place* features. These classes are chosen so that they collect together features

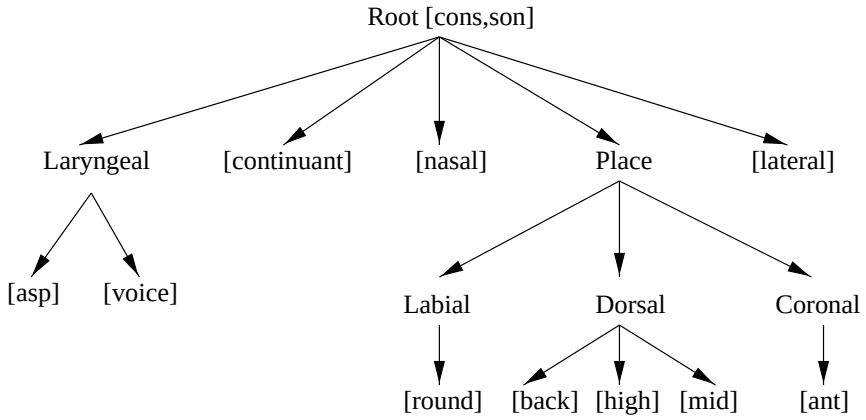

Figure 1: The feature geometry of Dutch

that behave as a unit in phonological processes of Dutch. The class of laryngeal features is basically making the voiced-voiceless distinction, while the ASPIRATION feature is only separating /h/ from the rest. The *place* class bundles together three subclasses of place of articulation features, one for each articulator. Furthermore some derived or redundant features such as GLIDE, APPROXIMANT and LIQUID are defined, but not shown in Figure 1. The vowels do not include the schwa, which is set apart and only specified as SCHWA+.

Using the Laplace evaluation function and this background knowledge, the constructed theory consisted of 13 prefix and 93 suffix rules, accepting 94.2% of the test positives and under 7.4% of the test negatives.

Among the rejected positives are loan words ('jeep' and 'junk' once again), but also all the words starting with perfectly Dutch /s/ - obstruent - liquid clusters.

The prefix rule with the widest coverage is:

```
prefix(A,B,C) :-
   head(C,D), sonorant(D,plu), rest(B,[]).
```

or, in other words, 'prefix anything before a single consonant before a nucleus other than the schwa'.

The suffix rules were less strict, with only 3 rejected positives, 'branche', 'dumps' and 'krimpst' (the first two of which are loan words) that failed to suffix /ʃ/, /s/ and /s/ respectively. Some achieve wide coverage (although never quite as wide as that of the prefix rules,) but some are making reference to individual phonemes and are of more restricted application. For example:

```
suffix(A,B,C) :-
   rest(C,D), head(D,E), rest(B,[]), A=t.
```

phoneme	obstruents	m	n	l	r	glides	vowels
sonority	1	2	2.25	2.5	2.75	3	4

Table 1: The Sonority Scale

or, 'suffix a /t/ after exactly one consonant, if the nucleus is a long vowel or a diphthong'.

Of some interest are also the end-of-word marking rules (see in section 4 above about the ^ mark), because of the fact that open, short monosyllables are very rare in Dutch (there are four in CELEX: 'schwa', 'ba', 'hè', and 'joh'). This would suggest that the best way to treat those is as exceptions, and have the general rule disallow open, short monosyllables. What was learned instead was a whole set of 29 rules for suffixing ^, the most general of which is:

```
suffix(A,B,C) :-
   head(B,t), larynx(t,E), rest(B,F),
   head(F,G), larynx(G,E), A= '^'.
```

or 'suffix an end-of-word mark after at least two consonants, if the outer-most one is a /t/ and has the same values for all the features in the LARYNGEAL feature class as the consonant immediately preceding it'.

A final note that needs to be made regarding this experiment, is one regarding its computational complexity. Overlapping and redundant features might be offering the opportunity for more interesting hypotheses, but are also making the search space bigger. The reason is that overlapping features are diminishing the effectiveness of the inverse resolution operator at keeping uninteresting predicates out of the bottom clause: the more background predicates can be used to prove the positive example on which the bottom clause is seeded, the longer the latter will get.

4.3 Sonority Scale

This last experiment implements and tests the syllabic structure model suggested by van der Hulst [9, ch. 3]. This is not a machine learning experiment, but a direct implementation of a hand-crafted syllable structure theory, and the presented for the sake of comparison. The Dutch syllable is analysed as having three prevocalic and 5 postvocalic positions (some of which may be empty), and constraints are placed on the set of consonants that can occupy each.

The most prominent constraint is the one stipulating a high-to-low *sonority* progression from the nucleus outwards. The theory is that each phoneme is assigned a sonority value (as in table 1) and syllables are then built from the nucleus outwards, by stacking segments of decreasing sonority.

The rough outline of the sonority scale is based on language-independent characteristics of the segments themselves and will, for example, place vowels higher than consonants and obstruents lower that any other consonant. The scale shown here for Dutch has been further refined in such a way that particular idiosyncrasies of the language are predicted.

So, for example, the nasals and the liquids are given in the original (language-independent) approximation of the sonority scale a sonority of 2. In the final, complete theory, however, they are evenly spread along the space between 2 and 3. The justification for this refinements is to allow for distinctions among the nasals and the liquids, so that, for example, /karl/ would be acceptable but /kalr/ not. It must, therefore, be noted that the prior knowledge of this theory is not only language-specific, but is also directly aimed at solving the very problem that is being investigated. In other words, it is a very informed piece of prior knowledge that, at least partially, encodes a hand-crafted model of Dutch syllable structure.

In addition to the high-to-low sonority level progression from the nucleus outwards, there are both *filters* and explicit licensing rules. The former are restrictions referring to sonority (e.g. 'the sonority of the three left-most positions must be smaller than 4') or other phonological features (e.g. the 'no voiced obstruents after the vowel' filter in p. 92) and are applicable in conjunction to the sonority rule. The latter are typically restricted in scope rules, that take precedence over the sonority-related constraints mentioned so far. The left-most position, for example, may be /s/ or empty, regardless of the contents of the rest of the prevocalic material.

Implementing the basic sonority progression rule as well as the most widely-applicable filters and rules[3] yielded impressive compression rates matched with results lying between those of the two previous experiments: 93.1% recall, 83.2% precision.

5 Conclusions and Further Work

The quantitative results from the machine learning experiments presented above are collected in table 2, together with those of Tjong Kim Sang and Nerbonne [8][4] and the results from the sonority scale experiment. Those last ones in particular, are listed for comparison's sake and as the logical end-point of the progression towards more language- and task-specific prior assumptions.

The first two columns are directly comparable, because they are both only referring to phonetic primitives with no linguistically sophisticated

[3]Some were left out because they were too lengthy when translated from their fixed-position framework to the affix licensing one used here, and were also very specifically fine tuning the theory to individual consonant clusters.

[4]From experiments on phonetic data in the 'Experiments without Linguistic Constraints' section.

	(Tjong 2000)	IPA	Feat. Classes	Sonority
Recall	99.1%	99.3%	94.2%	93.1%
Precision	74.8%	79.8%	92.6%	83.2%
Num. Clauses	$577 + 577$	$145 + 36$	$13 + 93$	3+8

Table 2: Results

background knowledge. The fact that the $\mathcal{C}_3\mathcal{V}\mathcal{C}_5$ template assumed in this work is not taken for granted in [8], is compensated in terms of compactness as well as performance. Compactness because the numbers of rules quoted in the first column do not include the 41 extra rules (besides the 1154 prefix and suffix rules) that describe the "basic word" on which the affix rules operate. Performance because the precision given is measured on totally random strings, whereas in this work only strings matching the $\mathcal{C}_3\mathcal{V}\mathcal{C}_5$ template are used.

As can be seen, then, the ILP-constructed rules compare favourably (in both performance and hypothesis compactness) with those constructed by the deductive approach employed in [8].

What can be also seen by comparing the two ILP results with each other, is that the drop in recall between the the second and third column is compensated by higher precision and compression, suggesting a direct correspondence between the quality of the prior knowledge encoded in the background theory and that of the constructed hypothesis.

The strongest argument in favour of ILP is that is employing background knowledge expressed in a symbolic formalism, allowing for easy transfer of prior domain knowledge into the process. This is consistent with the results shown presented here, since they demonstrate that the ILP algorithm is taking advantage of the more informed and complex background theory of feature classes to construct a better and shorter theory.

One interesting follow-up to these experiments would be attempting to expand their domain to that of syllables of polysyllabic words and, eventually, full word-forms. In the interest of keeping the problems of learning syllabic structure and syllable-boundary phonology apart, a way must be devised to derive from the positive data (i.e. a corpus of Dutch word-forms) examples for a distinct machine learning session for each task.

Furthermore, it would be interesting to carry out the same (or rather the equivalent) experiments on other parts of the CELEX corpus (e.g. English or German) and see to which extend the results-to-background relation follows the same patterns.

References

[1] Geert Booij. *The Phonology of Dutch*. The Phonology of the World's Languages. Clarendon Press, Oxford, 1995.

[2] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995. URL ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/InvEnt.ps.gz.

[3] Stephen Muggleton and Luc De Raedt. Inductive Logic Programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994. URL ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/lpj.ps.gz. Updated version of technical report CW 178, May 1993, Department of Computing Science, K.U. Leuven.

[4] John A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[5] Ashwin Srinivasan. *The Aleph Manual*. http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/, Last update: Nov 21, 2002.

[6] Ivilin Stoianov. *Connectionist Lexical Processing*. PhD thesis, Rijksuniversiteit Groningen, 2001. URL http://dissertations.ub.rug.nl/faculties/science/2001/i.p.stoianov/.

[7] Ivilin Stoianov and John Nerbonne. Exploring phonotactics with Simple Recurrent Networks. In van Eynde, Schuurman, and Schelkens, editors, *Proceedings of Computational Linguistics in the Netherlands 1998*, 1999.

[8] Erik F. Tjong Kim Sang and John Nerbonne. Learning the logic of simple phonotactics. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, volume 1925 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2000.

[9] Harry van der Hulst. *Syllable Structure and Stress in Dutch*. PhD thesis, Rijksuniversiteit Leiden, 1984.